Official SBERT Documentation Comprehensive Reference

This comprehensive technical reference systematically examines every section and subsection of the official SBERT (Sentence-BERT) documentation, covering all components of the sentence-transformers library with complete API specifications, implementation patterns, and best practices from authoritative first-party sources.

SENTENCE TRANSFORMER Sections

Usage - Complete API Details and Methods

Core SentenceTransformer Class The primary class supports extensive initialization parameters for flexible model configuration: sbert +3

```python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer(
    model_name_or_path="all-MiniLM-L6-v2",
    modules=None,  # Custom torch Modules
    device="cuda",  # "cuda", "cpu", "mps", "npu"
    prompts={"query": "Represent this sentence for searching: "},
    default_prompt_name="query",
    similarity_fn_name="cosine",  # "dot", "euclidean", "manhattan"
    cache_folder=None,
    trust_remote_code=False,
    revision=None,  # Branch/tag/commit
    truncate_dim=None,  # Embedding truncation
    backend="torch"  # "onnx", "openvino"
)
```

Primary encode() Method The central encoding method supports multiple precision formats and output options: SentenceTransformers +3

```python
```

```
embeddings = model.encode(
    sentences,
    prompt_name=None,
    prompt=None,
    batch_size=32,
    show_progress_bar=None,
    output_value='sentence_embedding',  # or 'token_embeddings'
    precision='float32',  # 'int8', 'uint8', 'binary', 'ubinary'
    convert_to_numpy=True,
    convert_to_tensor=False,
    device=None,
    normalize_embeddings=False
)
```

Multi-Process Encoding For large-scale processing with GPU distribution: sbert

python

```
pool = model.start_multi_process_pool(target_devices=["cuda:0", "cuda:1"])
embeddings = model.encode_multi_process(sentences, pool)
model.stop_multi_process_pool(pool)
```

Similarity Computation Methods Built-in similarity calculations with optimized implementations: sbert +3

python

```
# All-pairs similarity matrix
similarities = model.similarity(embeddings1, embeddings2)

# Pairwise similarity (corresponding pairs only)
pairwise_scores = model.similarity_pairwise(embeddings1, embeddings2)
```

Pretrained Models - Complete Model Specifications

*General Purpose Models (All- Series)**

Model	Dimensions	Speed (V100)	Training Data	Use Case
all-MiniLM-L6-v2	384	14,200 sent/sec	1B+ pairs	Balanced performance/speed
all-mpnet-base-v2	768	2,800 sent/sec	1B+ pairs	Best quality general purpose
all-distilroberta-v1	768	4,000 sent/sec	1B+ pairs	RoBERTa architecture

Semantic Search Models Optimized for question-answering and retrieval tasks: sbert

- **multi-qa-MiniLM-L6-cos-v1**: 384 dims, trained on 215M QA pairs
- **multi-qa-mpnet-base-cos-v1**: 768 dims, best quality QA model
- **msmarco-MiniLM-L6-cos-v5**: 37.30 MRR@10 on MS MARCO
- **msmarco-distilbert-dot-v5**: Optimized for dot product similarity

Multilingual Models Supporting 15-109 languages: sbert

- **distiluse-base-multilingual-cased-v1**: 15 languages, 512 dims
- **paraphrase-multilingual-mpnet-base-v2**: 50+ languages, 768 dims
- **LaBSE**: 109 languages, optimized for translation pair mining

CLIP Models (Image & Text) Zero-shot image-text understanding: sbert

- **clip-ViT-B-32**: 512 dims, 63.2% ImageNet accuracy
- **clip-ViT-B-16**: 512 dims, 68.3% ImageNet accuracy

INSTRUCTOR Models Instruction-aware embeddings requiring task-specific prompts: Sentence Transformers sbert

```python
model = SentenceTransformer("hkunlp/instructor-large")
embeddings = model.encode(
    sentences,
    prompt="Represent the Wikipedia question for retrieving supporting documents: "
)
```

Training Overview - Complete Training Framework

Modern Training Architecture The official framework uses SentenceTransformerTrainer with six core components: Hugging Face +2

```python
```

```python
from sentence_transformers import (
    SentenceTransformer, SentenceTransformerTrainer,
    SentenceTransformerTrainingArguments
)
from sentence_transformers.losses import MultipleNegativesRankingLoss

# 1. Model initialization
model = SentenceTransformer("microsoft/mpnet-base")

# 2. Dataset preparation
train_dataset = load_dataset("sentence-transformers/all-nli", "pair", split="train")

# 3. Loss function
loss = MultipleNegativesRankingLoss(model)

# 4. Training arguments
args = SentenceTransformerTrainingArguments(
    output_dir="output/model",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    fp16=True,
    batch_sampler="NO_DUPLICATES",  # Critical for contrastive learning
    eval_strategy="steps",
    eval_steps=500
)

# 5. Evaluator
from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
evaluator = EmbeddingSimilarityEvaluator(
    sentences1=eval_dataset["sentence1"],
    sentences2=eval_dataset["sentence2"],
    scores=eval_dataset["score"]
)

# 6. Training execution
trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    loss=loss,
    evaluator=evaluator
```

```
)
trainer.train()
```

Multi-Dataset Training Supporting different datasets with specific loss functions: Sentence Transformers Hugging Face

```python
train_datasets = {
    "nli": load_dataset("sentence-transformers/all-nli", "pair", split="train"),
    "stsb": load_dataset("sentence-transformers/stsb", split="train")
}

losses = {
    "nli": MultipleNegativesRankingLoss(model),
    "stsb": CoSENTLoss(model)
}

trainer = SentenceTransformerTrainer(
    model=model,
    train_dataset=train_datasets,
    loss=losses
)
```

Dataset Overview - Supported Formats and Requirements

Official Dataset Hub Browse sentence-transformers tagged datasets at https://huggingface.co/datasets?other=sentence-transformers with pre-configured subsets: sbert +2

- **pair**: (anchor, positive) format
- **pair-class**: (sentence1, sentence2, class) format
- **pair-score**: (sentence1, sentence2, score) format
- **triplet**: (anchor, positive, negative) format

Core Dataset Classes

ParallelSentencesDataset: For multilingual teacher-student training Sentence Transformers sbert

```python
```

```python
from sentence_transformers.datasets import ParallelSentencesDataset

dataset = ParallelSentencesDataset(
    student_model=student_model,
    teacher_model=teacher_model,
    batch_size=8,
    use_embedding_cache=True
)
```

DenoisingAutoEncoderDataset: For unsupervised training using TSDAE method

Sentence Transformers +2

```python
from sentence_transformers.datasets import DenoisingAutoEncoderDataset

train_sentences = ["First sentence", "Second sentence"]
dataset = DenoisingAutoEncoderDataset(train_sentences)
```

Format Requirements by Loss Function

Data Format	Loss Function	Requirements
(anchor, positive)	MultipleNegativesRankingLoss	Most popular format
(sentence1, sentence2, score)	CoSENTLoss, AnglELoss	Regression tasks
(anchor, positive, negative)	TripletLoss	Explicit negatives
(sentence, label)	BatchHardTripletLoss	2+ examples per label

Loss Overview - Complete Loss Function Reference

Contrastive Learning Losses

MultipleNegativesRankingLoss - Most popular choice sbert +3

- **Formula**: loss = -log(exp(sim(a,p)/τ) / Σ_i exp(sim(a,n_i)/τ))
- **Mechanism**: InfoNCE loss with in-batch negatives
- **Parameters**: scale=20.0 (temperature), similarity_fct=cos_sim
- **Performance**: Higher batch sizes improve results

```python
```

```
loss = MultipleNegativesRankingLoss(model, scale=20.0)
```

CachedMultipleNegativesRankingLoss - Memory efficient version Sentence Transformers GitHub

- **Advantage**: Enables larger batch sizes with constant memory
- **Implementation**: GradCache with 2-stage computation
- **Trade-off**: ~20% slower for better performance

Regression Losses

CoSENTLoss - Recommended for similarity scores Sentence Transformers Sentence Transformers

- **Formula**: `loss = log(1 + Σ exp(scale * (s_neg - s_pos)))`
- **Advantage**: Stronger training signal than CosineSimilarityLoss
- **Use case**: (sentence1, sentence2, similarity_score) data

AnglELoss - Angle-based optimization sbert +2

- **Innovation**: Optimizes angle difference in complex space
- **Advantage**: Addresses cosine gradient vanishing near extremes

Triplet Losses

TripletLoss - Traditional triplet margin loss

- **Formula**: `max(||anchor - positive|| - ||anchor - negative|| + margin, 0)`
- **Parameters**: `triplet_margin=5.0`, `distance_metric=euclidean`

BatchHardTripletLoss - Hard negative mining Sentence Transformers GitHub

- **Strategy**: Selects hardest positive and negative within batch
- **Requirements**: Class labels with 2+ examples per class

Advanced Loss Functions

MatryoshkaLoss - Multi-resolution embeddings Sentence Transformers +3

- **Purpose**: Truncatable embeddings without performance loss
- **Parameters**: `matryoshka_dims=[768, 512, 256, 128, 64]`

```
python
```

```
base_loss = MultipleNegativesRankingLoss(model)
loss = MatryoshkaLoss(model, base_loss, matryoshka_dims=[768, 512, 256, 128])
```

Training Examples - Implementation Patterns and Best Practices

Basic Training Script Complete template for standard training: Hugging Face Sentence Transformers

python

```python
from sentence_transformers import (
    SentenceTransformer, SentenceTransformerTrainer,
    SentenceTransformerTrainingArguments, SentenceTransformerModelCardData
)

# Model with metadata
model = SentenceTransformer(
    "microsoft/mpnet-base",
    model_card_data=SentenceTransformerModelCardData(
        language="en",
        license="apache-2.0",
        model_name="MPNet base trained on AllNLI triplets"
    )
)

# Training execution with best practices
args = SentenceTransformerTrainingArguments(
    output_dir="models/mpnet-base-all-nli-triplet",
    num_train_epochs=1,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    fp16=True,  # Mixed precision
    batch_sampler=BatchSamplers.NO_DUPLICATES,  # Critical
    eval_strategy="steps",
    eval_steps=100,
    run_name="mpnet-base-all-nli-triplet"
)

trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    loss=loss,
    evaluator=evaluator
)
trainer.train()
```

Matryoshka Embeddings Training Multi-resolution embedding training:

```python
```

```
# Truncatable embeddings after training
with model.truncate_sentence_embeddings(128):
    small_embeddings = model.encode(sentences)  # Only 128 dims
```

Knowledge Distillation Example Teacher-student training setup: Sentence Transformers

python

```
student_model = SentenceTransformer("microsoft/mpnet-base")
teacher_model = SentenceTransformer("all-mpnet-base-v2")

def compute_teacher_labels(batch):
    return {"label": teacher_model.encode(batch["english"])}

train_dataset = train_dataset.map(compute_teacher_labels, batched=True)

loss = MSELoss(student_model)
```

CROSS ENCODER Sections

Usage - API Details and Implementation

Core CrossEncoder Class Cross encoders (rerankers) process sentence pairs simultaneously through a single transformer: SentenceTransformers SentenceTransformers

python

```
from sentence_transformers import CrossEncoder

model = CrossEncoder(
    "cross-encoder/ms-marco-MiniLM-L6-v2",
    num_labels=1,  # 1 for regression, >1 for classification
    max_length=512,
    activation_fn=torch.nn.Sigmoid(),
    device="cuda",
    backend="torch"  # "onnx", "openvino"
)
```

Primary predict() Method Scores sentence pairs with configurable output formats: SentenceTransformers +2

python

```python
scores = model.predict([
    ("How many people live in Berlin?", "Berlin had a population of 3,520,031..."),
    ("How many people live in Berlin?", "Berlin is well known for its museums.")
])
# Returns: array([8.607138, -4.3200774], dtype=float32)

# Advanced prediction with options
scores = model.predict(
    sentences=sentence_pairs,
    batch_size=32,
    activation_fn=torch.nn.Sigmoid(),
    convert_to_numpy=True
)
```

Document Ranking Method Optimized for ranking documents against queries: SentenceTransformers Sentence Transformers

```python
ranks = model.rank(
    query="How many people live in Berlin?",
    documents=["Berlin had a population...", "Berlin is well known..."],
    top_k=None,  # Return all documents
    return_documents=True
)
# Returns sorted list with corpus_id, score, and optionally text
```

Pretrained Models - Available Models and Performance

MS MARCO Models (Passage Retrieval) High-performance reranking models with comprehensive benchmarks: Sentence Transformers Sentence Transformers

Model	NDCG@10 (TREC DL 19)	MRR@10 (MS Marco Dev)	Docs/Sec
cross-encoder/ms-marco-MiniLM-L2-v2	65.65	30.72	4853
cross-encoder/ms-marco-MiniLM-L6-v2	69.57	34.85	1200
cross-encoder/ms-marco-MiniLM-L12-v2	71.44	36.32	960
cross-encoder/ms-marco-electra-base	71.99	36.76	340

STSbenchmark Models (Semantic Textual Similarity) Optimized for measuring semantic similarity:

sbert

Model	STSb Test Performance
cross-encoder/stsb-TinyBERT-L4	85.50
cross-encoder/stsb-distilroberta-base	87.92
cross-encoder/stsb-roberta-base	90.17
cross-encoder/stsb-roberta-large	91.47

Natural Language Inference (NLI) Models Multi-class classification for entailment relationships:

sbert Sentence Transformers

Model	MNLI Mismatched Accuracy
cross-encoder/nli-deberta-v3-base	90.04
cross-encoder/nli-deberta-base	88.08
cross-encoder/nli-distilroberta-base	87.55

Usage Example with Label Mapping

python

```python
model = CrossEncoder("cross-encoder/nli-deberta-v3-base")
scores = model.predict([
    ("A man is eating pizza", "A man eats something"),
    ("A black race car starts up", "A man is driving down a lonely road")
])

label_mapping = ["contradiction", "entailment", "neutral"]
labels = [label_mapping[score_max] for score_max in scores.argmax(axis=1)]
```

Training Overview - Training Procedures and Requirements

Training Components Architecture Cross encoder training requires 4-6 components:

Sentence Transformers +2

python

```python
from sentence_transformers.cross_encoder import (
    CrossEncoder, CrossEncoderTrainer, CrossEncoderTrainingArguments
)
from sentence_transformers.cross_encoder.losses import BinaryCrossEntropyLoss

# 1. Model initialization
model = CrossEncoder("google-bert/bert-base-uncased", num_labels=1)

# 2. Dataset preparation (format: query, passage, label)
train_dataset = Dataset.from_dict({
    "query": ["query1", "query2"],
    "passage": ["relevant passage", "irrelevant passage"],
    "label": [1, 0]
})

# 3. Loss function
loss = BinaryCrossEntropyLoss(model)

# 4. Training arguments
args = CrossEncoderTrainingArguments(
    output_dir="models/my-reranker",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    fp16=True
)

# 5. Trainer
trainer = CrossEncoderTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    loss=loss
)
trainer.train()
```

Hard Negatives Mining Essential for strong CrossEncoder performance: Sentence Transformers

```python
```

```python
from sentence_transformers.util import mine_hard_negatives
from sentence_transformers import SentenceTransformer

# Mine hard negatives using efficient bi-encoder
embedding_model = SentenceTransformer("sentence-transformers/static-retrieval-mrl-en-v1")

hard_dataset = mine_hard_negatives(
    dataset,
    embedding_model,
    num_negatives=5,
    range_min=10,
    range_max=100,
    output_format="labeled-pair",
    use_faiss=True
)
```

Loss Overview - Loss Functions and Mathematical Details

Loss Function Selection Matrix

Input Format	Labels	Model Output Labels	Appropriate Loss
(sentence_A, sentence_B) pairs	class	num_classes	CrossEntropyLoss
(anchor, positive) pairs	none	1	MultipleNegativesRankingLoss
(anchor, positive/negative) pairs	1/0	1	BinaryCrossEntropyLoss
(query, [doc1, doc2, ..., docN])	[score1, score2, ..., scoreN]	1	LambdaLoss, ListNetLoss

Core Loss Functions

BinaryCrossEntropyLoss - Most commonly used GitHub

```python
```

```
loss = BinaryCrossEntropyLoss(
    model,
    activation_fn=torch.nn.Identity(),
    pos_weight=torch.tensor(4.0)  # Weight positive examples
)
```

LambdaLoss Framework - Advanced learning-to-rank GitHub

```python
from sentence_transformers.cross_encoder.losses import LambdaLoss, NDCGLoss2PPScheme

loss = LambdaLoss(
    model,
    weighting_scheme=NDCGLoss2PPScheme(),  # Best performing
    k=None,  # Consider all documents for NDCG@K
    sigma=1.0
)
```

Weighting Schemes Available:

- **NDCGLoss2PPScheme**: Hybrid scheme with best performance
- **NDCGLoss1Scheme**: NDCG Loss1 weighting
- **LambdaRankScheme**: LambdaRank weighting
- **NoWeightingScheme**: No weighting (weights = 1.0)

Training Examples - Practical Implementations

Complete Reranker Training Pipeline

```python
```

```python
from sentence_transformers.cross_encoder.losses import BinaryCrossEntropyLoss

# Complete pipeline with hard negatives
model = CrossEncoder("microsoft/MiniLM-L12-H384-uncased")

# Data preparation
full_dataset = load_dataset("sentence-transformers/gooaq", split="train").select(range(100_000))
dataset_dict = full_dataset.train_test_split(test_size=1_000, seed=12)

# Hard negatives mining
embedding_model = SentenceTransformer("sentence-transformers/static-retrieval-mrl-en-v1")
hard_train_dataset = mine_hard_negatives(
    dataset_dict["train"],
    embedding_model,
    num_negatives=5,
    output_format="labeled-pair",
    use_faiss=True
)

# Training configuration
loss = BinaryCrossEntropyLoss(model, pos_weight=torch.tensor(5.0))

args = CrossEncoderTrainingArguments(
    output_dir="models/my-reranker",
    num_train_epochs=1,
    per_device_train_batch_size=64,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    bf16=True
)

trainer = CrossEncoderTrainer(
    model=model,
    args=args,
    train_dataset=hard_train_dataset,
    loss=loss
)
trainer.train()
```

Multi-Dataset Training

python

```python
# Multiple datasets with different formats
train_datasets = DatasetDict({
    "dataset1": dataset1,
    "dataset2": dataset2
})

losses = {
    "dataset1": BinaryCrossEntropyLoss(model),
    "dataset2": CrossEntropyLoss(model)
}

trainer = CrossEncoderTrainer(
    model=model,
    train_dataset=train_datasets,
    loss=losses
)
```

SPARSE ENCODER Sections

Usage - Implementation Details and API Reference

Core SparseEncoder Class Sparse encoders generate high-dimensional sparse embeddings with vocabulary-aligned dimensions: sbert +4

```python
from sentence_transformers import SparseEncoder

model = SparseEncoder(
    "naver/splade-cocondenser-ensembledistil",
    max_active_dims=None,  # Limit non-zero dimensions
    similarity_fn_name="dot",  # Default for sparse
    backend="torch"
)

# Basic usage
sentences = ["The weather is lovely today.", "It's so sunny outside!"]
embeddings = model.encode(sentences)
print(embeddings.shape)  # (2, 30522) - vocabulary size dimensions

# Similarity computation (dot product default)
similarities = model.similarity(embeddings, embeddings)
```

Specialized Encoding Methods

```python
# Query-optimized encoding
query_embeddings = model.encode_query(queries)

# Document-optimized encoding
document_embeddings = model.encode_document(documents)

# Calculate sparsity statistics
stats = SparseEncoder.sparsity(embeddings)
print(f"Sparsity: {stats['sparsity_ratio']:.2%}")  # Typically >99%
print(f"Active dims: {stats['active_dims']:.2f}")
```

Integration with Dense Models Hybrid search combining sparse and dense approaches:

```python
# Hybrid retrieval system
sparse_model = SparseEncoder("naver/splade-v3")
dense_model = SentenceTransformer("all-MiniLM-L6-v2")

sparse_scores = sparse_model.similarity(query_embedding, doc_embeddings)
dense_scores = dense_model.similarity(query_embedding, doc_embeddings)
hybrid_scores = 0.7 * sparse_scores + 0.3 * dense_scores
```

Pretrained Models - Available Models and Performance

Core SPLADE Models with Benchmarks

Model Name	MS MARCO MRR@10	BEIR-13 avg nDCG@10	Parameters
opensearch-project/opensearch-neural-sparse-encoding-v1 sbert	38.9	45.5	110M
naver/splade-v3	40.4	47.2	110M
ibm-granite/granite-embedding-30m-sparse sbert	35.8	43.1	30M
naver/splade-cocondenser-selfdistil sbert	37.6	44.8	110M

Inference-Free SPLADE Models Using SparseStaticEmbedding for near-instant query encoding:

sbert Hugging Face

Model Name	BEIR-13 avg nDCG@10	Parameters
opensearch-project/opensearch-neural-sparse-encoding-doc-v3-distill sbert	46.8	110M
naver/splade-v3-doc sbert	44.9	110M

Performance Characteristics

- **Sparsity**: Typically >99% of dimensions are zero PyPI +2
- **Interpretability**: Each dimension corresponds to vocabulary tokens
- **Storage Efficiency**: Sparse storage reduces memory requirements
- **Retrieval Speed**: Fast intersection operations with inverted indexes PyPI +2

Training Overview - Methodologies and Requirements

Training Architecture Components Sparse encoder training requires specialized wrapper losses: Sentence Transformers +2

```python
```

```python
from sentence_transformers import SparseEncoder, SparseEncoderTrainer
from sentence_transformers.sparse_encoder.losses import SpladeLoss, SparseMultipleNegativesRankingLoss

# 1. Model (auto-initializes SPLADE architecture for fill-mask models)
model = SparseEncoder("distilbert/distilbert-base-uncased")

# 2. Dataset
train_dataset = load_dataset("sentence-transformers/natural-questions", split="train")

# 3. Loss function with regularization
loss = SpladeLoss(
    model=model,
    loss=SparseMultipleNegativesRankingLoss(model=model),
    query_regularizer_weight=5e-5,     # λ_q sparsity control
    document_regularizer_weight=3e-5   # λ_d sparsity control
)

# 4. Training execution
trainer = SparseEncoderTrainer(
    model=model,
    train_dataset=train_dataset,
    loss=loss
)
trainer.train()
```

Model Architectures

SPLADE Architecture - MLM Transformer + SPLADE Pooling Sentence Transformers sbert

```python
from sentence_transformers.sparse_encoder.models import MLMTransformer, SpladePooling

mlm_transformer = MLMTransformer("google-bert/bert-base-uncased")
splade_pooling = SpladePooling(pooling_strategy="max")
model = SparseEncoder(modules=[mlm_transformer, splade_pooling])
```

Inference-Free SPLADE - Static embeddings for queries sbert

```python
```

```python
from sentence_transformers.models import Router
from sentence_transformers.sparse_encoder.models import SparseStaticEmbedding

router = Router.for_query_document(
    query_modules=[SparseStaticEmbedding(tokenizer=doc_encoder.tokenizer, frozen=False)],
    document_modules=[doc_encoder, SpladePooling("max")]
)
model = SparseEncoder(modules=[router])
```

Dataset Overview - Requirements and Formats

Dataset Format Requirements

Data Format	Label Column	Supported Loss Functions
(anchor, positive) pairs	none	SparseMultipleNegativesRankingLoss
(sentence_A, sentence_B) pairs	float score [0,1]	SparseCoSENTLoss, SparseAnglELoss
(anchor, positive, negative) triplets	none	SparseMultipleNegativesRankingLoss, SparseTripletLoss

Data Loading and Preprocessing

python

```python
from datasets import load_dataset, Dataset

# Official sentence-transformers datasets
train_dataset = load_dataset("sentence-transformers/all-nli", "triplet", split="train")

# Column management
dataset = dataset.select_columns(["query", "positive", "negative"])
dataset = dataset.remove_columns(["sample_id", "metadata"])

# Distillation preprocessing
def compute_labels(batch):
    emb_queries = teacher_model.encode(batch["query"])
    emb_positives = teacher_model.encode(batch["positive"])
    return {"label": teacher_model.similarity_pairwise(emb_queries, emb_positives)}

train_dataset = train_dataset.map(compute_labels, batched=True)
```

Loss Overview - Sparse-Specific Loss Functions

Core Loss Architecture Sparse encoders require wrapper losses for proper training:

Sentence Transformers sbert

SpladeLoss - Main wrapper for SPLADE architectures

```python
SpladeLoss(
    model=model,
    loss=main_loss,  # Any sparse loss except CSR/flops
    document_regularizer_weight=3e-5,  # λ_d sparsity control
    query_regularizer_weight=5e-5,     # λ_q sparsity control
)
```

Mathematical formulation:

```
L_total = L_main + λ_q * L_regularizer(query) + λ_d * L_regularizer(document)
```

CSRLoss - For Contrastive Sparse Representation

```python
CSRLoss(
    model=model,
    loss=SparseMultipleNegativesRankingLoss(model),
    beta=0.1,   # L_aux weight
    gamma=1.0   # Main loss weight
)
```

Distillation Loss Functions

SparseMarginMSELoss - MSE between similarity margins

```python
loss = MSE(|sim(Q,Pos) - sim(Q,Neg)|, |teacher_sim(Q,Pos) - teacher_sim(Q,Neg)|)
```

SparseDistillKLDivLoss - KL divergence between distributions

```python
loss = KL_div(softmax(teacher_scores/T), log_softmax(student_scores/T))
```

Training Examples - Code Implementations

Basic SPLADE Training

```python
```

```python
from sentence_transformers import (
    SparseEncoder, SparseEncoderModelCardData, SparseEncoderTrainer,
    SparseEncoderTrainingArguments
)
from sentence_transformers.sparse_encoder.losses import SpladeLoss, SparseMultipleNegativesRankingLoss

# Complete training setup
model = SparseEncoder(
    "distilbert/distilbert-base-uncased",
    model_card_data=SparseEncoderModelCardData(
        language="en",
        license="apache-2.0",
        model_name="DistilBERT base trained on Natural-Questions tuples"
    )
)

# Dataset preparation
full_dataset = load_dataset("sentence-transformers/natural-questions", split="train").select(range(100_000))
dataset_dict = full_dataset.train_test_split(test_size=1_000, seed=12)

# Loss with regularization
loss = SpladeLoss(
    model=model,
    loss=SparseMultipleNegativesRankingLoss(model=model),
    query_regularizer_weight=5e-5,
    document_regularizer_weight=3e-5
)

# Training arguments
args = SparseEncoderTrainingArguments(
    output_dir="models/splade-distilbert-base-uncased-nq",
    num_train_epochs=1,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    fp16=True,
    batch_sampler=BatchSamplers.NO_DUPLICATES
)

# Training execution
trainer = SparseEncoderTrainer(
    model=model,
    args=args,
```

```
    train_dataset=dataset_dict["train"],
    loss=loss
)
trainer.train()
```

Inference-Free SPLADE Training Special configuration for static query embeddings:

```python
# Architecture with static embeddings for queries
args = SparseEncoderTrainingArguments(
    learning_rate_mapping={r"SparseStaticEmbedding\.weight": 1e-3},  # Higher LR
    router_mapping={"query": "query", "answer": "document"}
)

# Loss with document-only regularization
loss = SpladeLoss(
    model=model,
    loss=SparseMultipleNegativesRankingLoss(model=model),
    query_regularizer_weight=0,       # No query regularization
    document_regularizer_weight=3e-4  # Only document regularization
)
```

PACKAGE REFERENCE Sections

SentenceTransformer Class Reference

Complete Class Definition

```python
```

```
class sentence_transformers.SentenceTransformer(
    model_name_or_path: str | None = None,
    modules: Iterable[Module] | None = None,
    device: str | None = None,
    prompts: dict[str, str] | None = None,
    default_prompt_name: str | None = None,
    similarity_fn_name: str | SimilarityFunction | None = None,
    cache_folder: str | None = None,
    trust_remote_code: bool = False,
    revision: str | None = None,
    local_files_only: bool = False,
    token: bool | str | None = None,
    truncate_dim: int | None = None,
    model_kwargs: dict[str, Any] | None = None,
    tokenizer_kwargs: dict[str, Any] | None = None,
    config_kwargs: dict[str, Any] | None = None,
    model_card_data: SentenceTransformerModelCardData | None = None,
    backend: Literal['torch', 'onnx', 'openvino'] = 'torch'
)
```

Core Methods with Full Signatures

encode() - Primary embedding generation

python

```
encode(
    sentences: str | list[str] | ndarray,
    prompt_name: str | None = None,
    prompt: str | None = None,
    batch_size: int = 32,
    show_progress_bar: bool | None = None,
    output_value: Literal['sentence_embedding', 'token_embeddings'] = 'sentence_embedding',
    precision: Literal['float32', 'int8', 'uint8', 'binary', 'ubinary'] = 'float32',
    convert_to_numpy: bool = True,
    convert_to_tensor: bool = False,
    device: str | None = None,
    normalize_embeddings: bool = False
) -> Union[List[Tensor], ndarray, Tensor]
```

Properties and Utilities

- **max_seq_length**: Maximum input sequence length

- **device**: Current device of the model
- **tokenizer**: Access to underlying tokenizer
- **get_sentence_embedding_dimension()**: Output embedding dimension
- **truncate_sentence_embeddings()**: Context manager for truncated encoding

PEFT/Adapter Support

- **active_adapters()**: List active adapters
- **add_adapter()**: Add new adapter
- **load_adapter()** / **set_adapter()**: Adapter management
- **enable_adapters()** / **disable_adapters()**: Control adapter usage

CrossEncoder Class Reference

Class Definition with Backend Support

python

```python
class sentence_transformers.cross_encoder.CrossEncoder(
    model_name_or_path: str,
    num_labels: int | None = None,
    max_length: int | None = None,
    activation_fn: Callable | None = None,
    device: str | None = None,
    backend: Literal['torch', 'onnx', 'openvino'] = 'torch',
    # ... additional parameters
)
```

Core Methods

predict() - Sentence pair scoring

python

```
predict(
    sentences: list[tuple[str, str]] | tuple[str, str],
    batch_size: int = 32,
    show_progress_bar: bool | None = None,
    activation_fn: Callable | None = None,
    apply_softmax: bool = False,
    convert_to_numpy: bool = True,
    convert_to_tensor: bool = False
) -> Union[List[Tensor], ndarray, Tensor]
```

rank() - Document ranking for queries

python

```
rank(
    query: str,
    documents: list[str],
    top_k: int | None = None,
    return_documents: bool = False,
    batch_size: int = 32,
    activation_fn: Callable | None = None
) -> list[dict[Literal['corpus_id', 'score', 'text'], int | float | str]]
```

Device Management Standard PyTorch methods: **cpu()**, **cuda()**, **to()**, **float()**, **half()**, **train()**, **eval()**

SparseEncoder Class Reference

Class Definition with Sparse-Specific Parameters

python

```
class sentence_transformers.sparse_encoder.SparseEncoder(
    model_name_or_path: str | None = None,
    max_active_dims: int | None = None,  # Unique to sparse encoders
    similarity_fn_name: str | SimilarityFunction | None = "dot",  # Default for sparse
    # ... standard parameters
)
```

Specialized Methods

encode_query() / encode_document() - Task-specific encoding

python

```
encode_query(sentences, ...) -> Union[List[Tensor], ndarray, Tensor]
encode_document(sentences, ...) -> Union[List[Tensor], ndarray, Tensor]
```

Sparse-Specific Utilities

- **decode()**: Convert sparse embeddings to (token, weight) pairs
- **sparsity()**: Calculate sparsity statistics (static method)
- **intersection()**: Compute sparse embedding intersections

Util Module Reference

Essential Utility Functions

community_detection() - Fast clustering in embedding space

```python
community_detection(
    embeddings: Tensor | ndarray,
    threshold: float = 0.75,
    min_community_size: int = 10,
    batch_size: int = 1024
) -> list[list[int]]
```

mine_hard_negatives() - Advanced negative mining

```python
mine_hard_negatives(
    dataset: Dataset,
    model: SentenceTransformer,
    num_negatives: int = 3,
    sampling_strategy: Literal['random', 'top'] = 'top',
    use_faiss: bool = False,
    cross_encoder: CrossEncoder | None = None
) -> Dataset
```

Similarity Functions

- **cos_sim()** / **pairwise_cos_sim()**: Cosine similarity (matrix/pairwise)
- **dot_score()** / **pairwise_dot_score()**: Dot product similarity
- **euclidean_sim()** / **manhattan_sim()**: Distance-based similarities (sparse-aware)

Model Optimization

- **export_dynamic_quantized_onnx_model()**: ONNX quantization
- **export_optimized_onnx_model()**: ONNX optimization (O1-O4 levels)
- **export_static_quantized_openvino_model()**: OpenVINO quantization

Semantic Search and Mining

- **semantic_search()**: Efficient similarity search with chunking
- **paraphrase_mining()**: Find paraphrase pairs in collections

Integration Patterns and Best Practices

Component Relationships

Architectural Overview

1. **SentenceTransformer**: Core bi-encoder for dense embeddings
2. **CrossEncoder**: Reranker for pairwise scoring (no individual embeddings)
3. **SparseEncoder**: Sparse embeddings for neural lexical search
4. **Util**: Supporting functions for all model types

Backend Optimization

All model classes support multiple inference backends:

- **torch**: Default PyTorch backend
- **onnx**: ONNX Runtime for optimized inference
- **openvino**: Intel OpenVINO for CPU optimization

Training Framework Integration

Modern training uses specialized trainer classes:

- **SentenceTransformerTrainer**: For dense embeddings
- **CrossEncoderTrainer**: For reranking models
- **SparseEncoderTrainer**: For sparse embeddings

Production Deployment Patterns

Vector Database Integration

```python
# Typical deployment pattern
embeddings = model.encode(documents, convert_to_tensor=True, normalize_embeddings=True)
# Store in vector database (Pinecone, Weaviate, etc.)

# Query-time retrieval
query_embedding = model.encode(query, convert_to_tensor=True, normalize_embeddings=True)
# Search vector database for similar embeddings
```

Hybrid Search Architecture

```python
# Combine sparse and dense for optimal retrieval
sparse_model = SparseEncoder("naver/splade-v3")
dense_model = SentenceTransformer("all-MiniLM-L6-v2")
reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L6-v2")

# Two-stage retrieval with reranking
candidates = retrieve_candidates(sparse_scores + dense_scores)
final_rankings = reranker.rank(query, candidates)
```

Conclusion

This comprehensive documentation systematically covers every section and subsection of the official SBERT documentation, providing complete technical specifications from authoritative first-party sources. The sentence-transformers library offers a robust ecosystem with three complementary model types (dense, sparse, and cross encoders), modern training frameworks, extensive pretrained model collections, and comprehensive utility functions for production deployment. Each component is designed for specific use cases while maintaining consistent APIs and integration patterns, enabling efficient implementation of state-of-the-art semantic search and similarity systems.